

Python IP Class Notebook

Nebhrajani A. V.

Contents

1	Preamble	2
1.1	Notebook Conventions	2
1.2	Hardware and Software Used	2
1.3	Acknowledgements	2
2	NumPy	3
2.1	Worksheet 2020-07-26	3
3	Pandas	8
3.1	Series	8
3.2	DataFrame	15
4	Matplotlib	30
4.1	Some Simple Plots	30

1 Preamble

1.1 Notebook Conventions

All code in this notebook is in Python unless specified otherwise. All code is syntax-highlighted, placed in boxes, and is line numbered. The output of the interpreter on `stdout` is printed directly below it, *verbatim*, thus.

```
1  # Print Hello world!
2  print("Hello world!")
```

Hello world!

It is recommended that you navigate using the hyperlinked TOC or the Adobe Bookmarks tree.

1.2 Hardware and Software Used

This notebook is written in an `org-mode` file and exported to PDF via L^AT_EX, Org version 9.3.6 on GNU Emacs 25.2.2 (x86_64-pc-linux-gnu, GTK+ Version 3.22.21) of 2017-09-23, modified by Debian, on a Foxconn Core i7 NanoPC running Linux Mint 19.3 XFCE 64-bit. Python 2.7.17 of 2020-04-15 is used throughout unless specified otherwise. For the Org or L^AT_EX source, contact aditya.v.nebhrajani@gmail.com.

1.3 Acknowledgements

I am grateful to the FSF, the GNU Project, the Linux foundation, the Emacs, StackExchange and FLOSS communities, and my father, who taught me that a world outside commercialized technology does exist and thrive.

2 NumPy

2.1 Worksheet 2020-07-26

1. Create an ndarray with values ranging from 10 to 49 each spaced with a difference of 3.

```
1 import numpy as np
2 arr=np.arange(10,50,3,dtype=int)
3 print(arr)
```

[10 13 16 19 22 25 28 31 34 37 40 43 46 49]

2. Find the output of the following Python code:

```
1 x="hello world"
2 print(x[:2],x[:-2],x[-2:])
```

he hello wor ld

3. Predict the output of the following code fragments:

```
1 import numpy as np
2 x=np.array([1,2,3])
3 y=np.array([3,2,1])
4 z=np.concatenate([x,y])
5 print(z)
```

[1 2 3 3 2 1]

4. Consider following two arrays: Array1= array([0,1,2],[3,4,5],[6,7,8]) and Array2= array([10,11,12],[13,14,15],[16,17,18]). Write NumPy command to concatenate Array1 and Array2:

(a) Row wise

```
1 import numpy as np
2 Array1= np.array([[0,1,2],[3,4,5],[6,7,8]])
3 Array2= np.array([[10,11,12],[13,14,15],[16,17,18]])
4 rarr=np.concatenate([Array1,Array2],axis=1)
5 print(rarr)
```

```
[[ 0  1  2 10 11 12]
 [ 3  4  5 13 14 15]
 [ 6  7  8 16 17 18]]
```

(b) Column wise

```
1  import numpy as np
2  Array1= np.array([[0,1,2],[3,4,5],[6,7,8]])
3  Array2= np.array([[10,11,12],[13,14,15],[16,17,18]])
4  carr=np.concatenate([Array1,Array2],axis=0)
5  print(carr)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [10 11 12]
 [13 14 15]
 [16 17 18]]
```

5. To create sequences of numbers, NumPy provides a function (a)arange analogous to range that returns arrays instead of lists.
6. Find the output of following program.

```
1  import numpy as np
2  a=np.array([30,60,70,30,10,86,45])
3  print(a[-2:6])
```

[86]

7. Write a NumPy program to create a 2d array with 1 on the border and 0 inside.

```
1  import numpy as np
2  x = np.ones((5,5))
3  print("Original array:")
4  print(x)
5  print("1 on the border and 0 inside in the array")
6  x[1:-1,1:-1] = 0
7  print(x)
```

Original array:

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
```

```

[1.  1.  1.  1.  1.]
1 on the border and 0 inside in the array
[[1.  1.  1.  1.  1.]
 [1.  0.  0.  0.  1.]
 [1.  0.  0.  0.  1.]
 [1.  0.  0.  0.  1.]
 [1.  1.  1.  1.  1.]]

```

8. Given following ndarray A: ([[2, 4, 6], [7, 8, 9], [1, 2, 3]]) Write the python statements to perform the array slices in the way so as to extract first row and second column.

```

1  import numpy as np
2  A = np.array([[2,4,6],[7,8,9],[1,2,3]])
3  print(A[0,:])
4  print(A[:,1])

```

```

[2 4 6]
[4 8 2]

```

9. Write python statement to create a two- dimensional array of 4 rows and 3 columns. The array should be filled with ones.

```

1  import numpy as np
2  x = np.ones((4,3))
3  print(x)

```

```

[[1.  1.  1.]
 [1.  1.  1.]
 [1.  1.  1.]
 [1.  1.  1.]]

```

10. Find the output of following program.

```

1  import numpy as np
2  d = np.array([10,20,30,40,50,60,70])
3  print(d[-5:])

```

```

[30 40 50 60 70]

```

11. State at least two differences between a NumPy array and a list

NumPy Array	List
By default, numpy arrays are homogeneous	They can have elements of different data types
Element-wise operations are possible	Element-wise operations don't work on lists
They take up less space	They take up more space

12. Find the output of following program.

```

1  import numpy as np
2  d=np.array([10,20,30,40,50,60,70])
3  print(d[-1:-4:-1])

```

[70 60 50]

13. Write the output of the following code.

```

1  import numpy as np
2  a=[[1,2,3,4],[5,6,7,8]]
3  b=[[1,2,3,4],[5,6,7,8]]
4  n=np.concatenate((a, b), axis=0)
5  print(n[1])
6  print(n[1][1])

```

[5 6 7 8]

6

14. Which of the following is contained in NumPy library?

- (a) **N-Dimensional Array Object**
- (b) Series
- (c) DataFrame
- (d) Plot

15. Point out the correct statement:

- (a) NumPy main object is the homogeneous multidimensional array
- (b) In Numpy, dimensions are called axes
- (c) NumPy array class is called ndarray
- (d) **All of the above**

16. When the fromiter() is preferred over array()? **A:** Fromiter() is preferred over array() for creating non-numeric sequences like strings and dictionaries.

17. What is the purpose of order argument in empty(). What do 'C' and 'F' stands for? What is the default value of order argument? **A:** The "order" argument arranges the elements of the array row-wise or column-wise. C order arranges elements column wise and means "c"-like, whereas F order arranges elements row wise and means "fortran"-like. Default value of order argument is C.
18. Differentiate split() from hsplit() and vsplit(). **A:** Split() function is a general function which can be used to split an array in numpy both horizontally and vertically by providing an axis. If the axis is 0 it is the same as hsplit() and if the axis is 1 it behaves as vsplit(). The difference between split() and hsplit(), vsplit() is that split() allows you to specify the axis that you wish, and hsplit() and vsplit() are for specific axes.
19. Find the output:

```
(a) import numpy as np
2    a = np.linspace(2.5,5,6)
3    print(a)
```

```
[2.5 3.   3.5 4.   4.5 5. ]
```

```
(b) import numpy as np
2    a=np.array([[0,2,4,6],[8,10,12,14],[16,18,20,22],[24,26,28,30]])
3    print(a)
4    print(a[:3,3:])
5    print(a[1::2,:3])
6    print(a[-3:-1,-4::2])
7    print(a[:, :-1, ::-1])
```

```
[[ 0  2  4  6]
 [ 8 10 12 14]
 [16 18 20 22]
 [24 26 28 30]]
[[ 6]
 [14]
 [22]]
[[ 8 10 12]
 [24 26 28]]
[[ 8 12]
 [16 20]]
[[30 28 26 24]
 [22 20 18 16]
 [14 12 10  8]
 [ 6  4  2  0]]
```

3 Pandas

3.1 Series

```
1  # Import numpy and pandas
2  import pandas as pd
3  import numpy as np
4
5  # Create an empty series
6  s = pd.Series()
7  print(s)
8
9  # Series from ndarray
10 data = np.array(['a', 'b', 'c', 'd'])
11
12 ## Without index
13 s = pd.Series(data)
14 print(s)
15 ## With index
16 s = pd.Series(data, index = [100, 101, 102, 103])
17 print(s)
18
19 # Scalar series
20 s = pd.Series(5, index = [0, 1, 2, 3])
21 print(s)
22
23 # Series from dictionary
24 data = {'a' : 0., 'b' : 1., 'c' : 2.}
25
26 ## Without index
27 s = pd.Series(data)
28 print(s)
29 ## With index
30 s = pd.Series(data, index = ['b', 'c', 'd', 'a'])
31 print(s)
32
33 # Another dictionary example
34 f_dict = {'apples': 500, 'kiwi': 20, 'oranges': 100, 'cherries': 6000}
35 print(f_dict)
36
37 arr = pd.Series(f_dict)
38 print('\nArray Items')
39 print(arr)
```

```
Series([], dtype: float64)
0      a
1      b
```



```

2      c
3      d
dtype: object
100    a
101    b
102    c
103    d
dtype: object
0      5
1      5
2      5
3      5
dtype: int64
a      0.0
b      1.0
c      2.0
dtype: float64
b      1.0
c      2.0
d      NaN
a      0.0
dtype: float64
{'apples': 500, 'kiwi': 20, 'oranges': 100, 'cherries': 6000}

```

```

Array Items
apples      500
kiwi        20
oranges     100
cherries    6000
dtype: int64

```

```

1  # Indexing
2  import pandas as pd
3  from pandas import Series
4  arr = Series([22, 44, 66, 88, 108])
5  print(arr[[1, 3, 0, 4]])

```

```

1      44
3      88
0      22
4     108
dtype: int64

```

```

1  # Series operations
2  import pandas as pd
3  ds1 = pd.Series([2, 4, 6, 8, 10])

```

```

4 ds2 = pd.Series([1, 3, 5, 7, 9])
5 print(ds1)
6 print(ds2)
7 ds = ds1 + ds2
8 print("Add two Series:")
9 print(ds)
10 print("Subtract two Series:")
11 ds = ds1 - ds2
12 print(ds)
13 print("Multiply two Series:")
14 ds = ds1 * ds2
15 print(ds)
16 print("Divide Series1 by Series2:")
17 ds = ds1 / ds2
18 print(ds)

```

```

0    2
1    4
2    6
3    8
4   10

```

dtype: int64

```

0    1
1    3
2    5
3    7
4    9

```

dtype: int64

Add two Series:

```

0    3
1    7
2   11
3   15
4   19

```

dtype: int64

Subtract two Series:

```

0    1
1    1
2    1
3    1
4    1

```

dtype: int64

Multiply two Series:

```

0    2
1   12
2   30
3   56

```

```
4      90
dtype: int64
Divide Series1 by Series2:
0      2.000000
1      1.333333
2      1.200000
3      1.142857
4      1.111111
dtype: float64
```

```
1  # Series to array
2  import pandas as pd
3  import numpy as np
4  s1 = pd.Series(['100', '200', '300', 'python'])
5  print("Original data series")
6  print(s1)
7  print("Series to array")
8  a = np.array(s1.values.tolist())
9  print(a)
```

```
Original data series
0      100
1      200
2      300
3      python
dtype: object
Series to array
['100' '200' '300' 'python']
```

```
1  # Heads and tails
2  import pandas as pd
3  import math
4  s = pd.Series(data = [math.sqrt(x) for x in range(1,10)],
5                  index = [x for x in range(1,10)])
6  print(s)
7  print(s.head(6))
8  print(s.tail(7))
9  print(s.head())
10 print(s.tail())
```

```
1      1.000000
2      1.414214
3      1.732051
4      2.000000
5      2.236068
```

```

6      2.449490
7      2.645751
8      2.828427
9      3.000000
dtype: float64
1      1.000000
2      1.414214
3      1.732051
4      2.000000
5      2.236068
6      2.449490
dtype: float64
3      1.732051
4      2.000000
5      2.236068
6      2.449490
7      2.645751
8      2.828427
9      3.000000
dtype: float64
1      1.000000
2      1.414214
3      1.732051
4      2.000000
5      2.236068
dtype: float64
5      2.236068
6      2.449490
7      2.645751
8      2.828427
9      3.000000
dtype: float64

```

```

1  # Sorting pandas series
2  import pandas as pd
3  s = pd.Series(['100', '200', 'python', '300.12', '400'])
4  print("Original data series:")
5  print(s)
6  asc_s = pd.Series(s).sort_values()
7  print(asc_s)
8  dsc_s = pd.Series(s).sort_values(ascending=False)
9  print(dsc_s)
10
11 # Appending
12 new_s = s.append(pd.Series(['500', 'php']))
13 print(new_s)

```

Original data series:

```
0      100
1      200
2    python
3    300.12
4      400
dtype: object
0      100
1      200
3    300.12
4      400
2    python
dtype: object
2    python
4      400
3    300.12
1      200
0      100
dtype: object
0      100
1      200
2    python
3    300.12
4      400
0      500
1      php
dtype: object
```

```
1  # Mean and median
2  import pandas as pd
3  s = pd.Series(data = [1,2,3,4,5,6,7,8,9,5,3])
4  print("Original data series:")
5  print(s)
6  print("Mean:")
7  print(s.mean())
8  print("Standard deviation:")
9  print(s.std())
```

Original data series:

```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
```

```
8      9
9      5
10     3
dtype: int64
Mean:
4.818181818181818
Standard deviation:
2.522624895547565
```

```
1  # Isin function
2  import numpy as np
3  import pandas as pd
4
5  s = pd.Series(['dog', 'cow', 'dog', 'cat', 'lion'], name='animal')
6
7  r = s.isin(['dog', 'cat'])
8  print(r)
```

```
0      True
1     False
2      True
3      True
4     False
Name: animal, dtype: bool
```

```
1  # Appending and concatenation
2  import numpy as np
3  import pandas as pd
4
5  # Input
6  ser1 = pd.Series(range(5))
7  ser2 = pd.Series(list('abcde'))
8
9  # Vertical
10 ser3 = ser1.append(ser2)
11 print(ser3)
12
13 # Or using Pandas concatenate along axis 0
14 ser3 = pd.concat([ser1, ser2], axis = 0)
15 print(ser3)
16
17 # Horizontal (into a dataframe)
18 ser3 = pd.concat([ser1, ser2], axis = 1)
19 print(ser3)
```

3.2 DataFrame

```
1  # Empty dataframe
2  import pandas as pd
3
4  data = pd.DataFrame()
5  print(data)
```

Empty DataFrame

Columns: []

Index: []

```
1  # Dataframe from list
2  import pandas as pd
3
4  table = [1, 2, 3, 4, 5]
5  data = pd.DataFrame(table)
6  print(data)
```

```
0
0 1
1 2
2 3
3 4
4 5
```

```
1  # Dataframe from mixed list
2  import pandas as pd
3
4  table = [[1, 'Nebhrajani'], [2, 'Python'], [3, 'Hello']]
5  data = pd.DataFrame(table)
6  print(data)
```

```
0      1
0 1  Nebhrajani
1 2      Python
2 3      Hello
```

```
1  # Column labels
2  import pandas as pd
3
4  table = [[1, 'Nebhrajani'], [2, 'Python'], [3, 'Hello']]
5  data = pd.DataFrame(table, columns = ['S.No', 'Name'])
6  print(data)
```

	S.No	Name
0	1	Nebhrajani
1	2	Python
2	3	Hello

```

1  # Random numbers dataframe
2  import numpy as np
3  import pandas as pd
4
5  d_frame = pd.DataFrame(np.random.randn(8, 4))
6  print(d_frame)

```

	0	1	2	3
0	1.229859	0.276771	-0.045632	0.259311
1	0.074277	-0.168431	1.667804	-1.407468
2	0.237926	-0.839273	-0.507893	-1.479707
3	-1.406518	0.925847	1.197716	-1.009840
4	-2.363665	-1.150008	0.667568	-0.633510
5	0.145881	0.369782	0.261877	-0.740075
6	0.807986	1.199728	-0.492667	0.222350
7	-0.043676	0.477135	0.145498	0.965713

```

1  # Dataframe from dict
2  import pandas as pd
3
4  table = {'name': ['Aditya', 'Aryan', 'Nebhrajani', 'Sahej'],
5           'Salary': [1000000, 1200000, 900000, 1100000]}
6
7  data = pd.DataFrame(table)
8  print(data)

```

	name	Salary
0	Aditya	1000000
1	Aryan	1200000
2	Nebhrajani	900000
3	Sahej	1100000

```

1  # Dataframe from some given dictionary data
2  import pandas as pd
3  import numpy as np
4
5  exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James',
6                        'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
7               'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
8               'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

```



```

9         'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes',
10                    'no', 'no', 'yes']}
11 labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
12
13 df = pd.DataFrame(exam_data , index=labels)
14 print(df)

```

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Matthew	14.5	1	yes
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

```

1  # Messing with columns
2  import pandas as pd
3
4  table = {'name': ['Aditya', 'Aryan', 'Nebhrajani', 'Sahej'],
5           'Age': [25, 32, 30, 26],
6           'Profession': ['Developer', 'Analyst', 'Admin', 'HR'],
7           'Salary': [1000000, 1200000, 900000, 1100000]
8         }
9
10 data1 = pd.DataFrame(table)
11 print(data1)
12
13 print('\n After Changing the Column Order')
14 data2 = pd.DataFrame(table, columns = ['name', 'Profession', 'Salary',
15                                       'Age'])
16 print(data2)
17 print('\n Using Wrong Column ')
18 data3 = pd.DataFrame(table, columns = ['name', 'Qualification',
19                                       'Salary',
20                                       'Age'])
21 print(data3)

```

	name	Age	Profession	Salary
0	Aditya	25	Developer	1000000
1	Aryan	32	Analyst	1200000
2	Nebhrajani	30	Admin	900000
3	Sahej	26	HR	1100000

After Changing the Column Order

	name	Profession	Salary	Age
0	Aditya	Developer	1000000	25
1	Aryan	Analyst	1200000	32
2	Nebhrajani	Admin	900000	30
3	Sahej	HR	1100000	26

Using Wrong Column

	name	Qualification	Salary	Age
0	Aditya	NaN	1000000	25
1	Aryan	NaN	1200000	32
2	Nebhrajani	NaN	900000	30
3	Sahej	NaN	1100000	26

```
1  # Dataframe indexing
2  import pandas as pd
3
4  table = {'name': ['Aditya', 'Aryan', 'Nebhrajani', 'Sahej'],
5           'Age': [25, 32, 30, 26],
6           'Profession': ['Developer', 'Analyst', 'Admin', 'HR'],
7           'Salary': [1000000, 1200000, 900000, 1100000]}
8
9  data = pd.DataFrame(table)
10 print(data)
11
12 print('\nSetting name as an index')
13 new_data = data.set_index('name')
14 print(new_data)
15
16 print('\nReturn Index Aditya Details')
17 print(new_data.loc['Aditya'])
```

	name	Age	Profession	Salary
0	Aditya	25	Developer	1000000
1	Aryan	32	Analyst	1200000
2	Nebhrajani	30	Admin	900000
3	Sahej	26	HR	1100000

Setting name as an index

	Age	Profession	Salary
name			
Aditya	25	Developer	1000000
Aryan	32	Analyst	1200000
Nebhrajani	30	Admin	900000
Sahej	26	HR	1100000

```
Return Index Aditya Details
Age          25
Profession   Developer
Salary       1000000
Name: Aditya, dtype: object
```

```
1  # Getting columns
2  import pandas as pd
3
4  table = {'name': ['Aditya', 'Aryan', 'Nebhrajani', 'Sahej'],
5           'Age': [25, 31, 35, 26],
6           'Salary': [100000, 120000, 700000, 110000]
7           }
8
9  data = pd.DataFrame(table)
10 print(data)
11 print('\nShape and Size of a DataFrame')
12 print(data.shape)
13 data2 = pd.DataFrame(table, columns = ['name', 'Profession', 'Salary',
14                                       'Age'])
15 data3 = pd.DataFrame(table, columns = ['name', 'Qualification',
16                                       'Salary',
17                                       'Age'])
18 print('Data2 Values ')
19 print(data2.values)
20 print('\nData3 Values ')
21 print(data3.values)
22 data1 = pd.DataFrame(table)
23 table = {'Age': [25, 32, 30, 26],
24          'Salary': [1000000, 1200000, 900000, 1100000]
25          }
26 data4 = pd.DataFrame(table)
27 data1.index.name = 'Emp No'
28 print(data1)
29 print()
30 data4.index.name = 'Cust No'
31 print(data4)
32 data1.columns.name = 'Employee Details'
33 print(data1)
34 data4.columns.name = 'Customers Information'
35 print(data4)
36 data1 = pd.DataFrame(table)
37 print(data1)
38 print('\nDescribe function result')
39 print(data1.describe())
```

```
name  Age  Salary
```

0	Aditya	25	100000
1	Aryan	31	120000
2	Nebhrajani	35	700000
3	Sahej	26	110000

Shape and Size of a DataFrame

(4, 3)

Data2 Values

```

[['Aditya' nan 100000 25]
 ['Aryan' nan 120000 31]
 ['Nebhrajani' nan 700000 35]
 ['Sahej' nan 110000 26]]

```

Data3 Values

```

[['Aditya' nan 100000 25]
 ['Aryan' nan 120000 31]
 ['Nebhrajani' nan 700000 35]
 ['Sahej' nan 110000 26]]

```

	name	Age	Salary
Emp No			
0	Aditya	25	100000
1	Aryan	31	120000
2	Nebhrajani	35	700000
3	Sahej	26	110000

	Age	Salary
Cust No		
0	25	1000000
1	32	1200000
2	30	900000
3	26	1100000

Employee Details	name	Age	Salary
Emp No			
0	Aditya	25	100000
1	Aryan	31	120000
2	Nebhrajani	35	700000
3	Sahej	26	110000

Customers Information	Age	Salary
Cust No		
0	25	1000000
1	32	1200000
2	30	900000
3	26	1100000

	Age	Salary
0	25	1000000
1	32	1200000
2	30	900000
3	26	1100000

Describe function result

	Age	Salary
count	4.000000	4.000000e+00
mean	28.250000	1.050000e+06
std	3.304038	1.290994e+05
min	25.000000	9.000000e+05
25%	25.750000	9.750000e+05
50%	28.000000	1.050000e+06
75%	30.500000	1.125000e+06
max	32.000000	1.200000e+06

```
1  # Getting rows using loc
2  import pandas as pd
3  table = {'name': ['Jai', 'Mike', 'Suresh', 'Sahej'],
4           'Age': [25, 32, 30, 26],
5           'Profession': ['Developer', 'Analyst', 'Admin', 'HR'],
6           'Salary': [1000000, 1200000, 900000, 1100000]}
7
8  data = pd.DataFrame(table, index = ['a', 'b', 'c', 'd'])
9  print(data)
10
11 print('\n---Select b row from a DataFrame---')
12 print(data.loc['b'])
13
14 print('\n---Select c row from a DataFrame---')
15 print(data.loc['c'])
16
17 print('\n---Select b and d rows from a DataFrame---')
18 print(data.loc[['b', 'd']])
```

	name	Age	Profession	Salary
a	Jai	25	Developer	1000000
b	Mike	32	Analyst	1200000
c	Suresh	30	Admin	900000
d	Sahej	26	HR	1100000

---Select b row from a DataFrame---

name	Mike
Age	32
Profession	Analyst
Salary	1200000

Name: b, dtype: object

---Select c row from a DataFrame---

name	Suresh
Age	30

```

Profession      Admin
Salary          900000
Name: c, dtype: object

```

---Select b and d rows from a DataFrame---

```

      name Age Profession  Salary
b  Mike   32    Analyst 1200000
d  Sahej   26         HR 1100000

```

```

1  # Getting columns using loc
2  import pandas as pd
3  table = {'Name': ['Abhimanyu', 'Jai', 'Suresh', 'Sahej', 'Shail'],
4           'Age': [35, 25, 32, 30, 29],
5           'Profession': ['Manager', 'Developer', 'Analyst', 'Admin',
6                          'HR'],
7           'Sale': [422.19, 22.55, 119.47, 200.19, 44.55],
8           'Salary': [12000, 10000, 14000, 11000, 14000]}
9
10 data = pd.DataFrame(table)
11 print(data)
12
13 print('\n---Select Name, Sale column in a DataFrame---')
14 print(data.loc[:, ['Name', 'Sale']])
15
16 print('\n---Select Name, Profession, Salary in a DataFrame---')
17 print(data.loc[:, ['Name', 'Profession', 'Salary']])
18
19 print('\n---Select rows from 1 to 2 in a DataFrame---')
20 print(data.loc[1:3, ['Name', 'Profession', 'Salary']])

```

```

      Name Age Profession  Sale  Salary
0  Abhimanyu   35    Manager 422.19   12000
1         Jai   25  Developer  22.55   10000
2    Suresh   32    Analyst 119.47   14000
3     Sahej   30     Admin  200.19   11000
4     Shail   29         HR   44.55   14000

```

---Select Name, Sale column in a DataFrame---

```

      Name  Sale
0  Abhimanyu 422.19
1         Jai  22.55
2    Suresh 119.47
3     Sahej 200.19
4     Shail  44.55

```

---Select Name, Profession, Salary in a DataFrame---

```

      Name Profession  Salary

```

0	Abhimanyu	Manager	12000
1	Jai	Developer	10000
2	Suresh	Analyst	14000
3	Sahej	Admin	11000
4	Shail	HR	14000

---Select rows from 1 to 2 in a DataFrame---

	Name	Profession	Salary
1	Jai	Developer	10000
2	Suresh	Analyst	14000
3	Sahej	Admin	11000

```

1  # Getting rows using iloc
2  import pandas as pd
3
4  table = {'name': ['Jai', 'Mit', 'Suresh', 'Tammanah'],
5           'Age': [25, 32, 30, 26],
6           'Profession': ['Developer', 'Analyst', 'Admin', 'HR'],
7           'Salary': [1000000, 1200000, 900000, 1100000]}
8  data = pd.DataFrame(table, index = ['a', 'b', 'c', 'd'])
9  print(data)
10
11 print('\n---Select 1st row from a DataFrame---')
12 print(data.iloc[1])
13
14 print('\n---Select 3rd row from a DataFrame---')
15 print(data.iloc[3])
16
17 print('\n---Select 1 and 3 rows from a DataFrame---')
18 print(data.iloc[[1, 3]])

```

	name	Age	Profession	Salary
a	Jai	25	Developer	1000000
b	Mit	32	Analyst	1200000
c	Suresh	30	Admin	900000
d	Tammanah	26	HR	1100000

---Select 1st row from a DataFrame---

name	Mit
Age	32
Profession	Analyst
Salary	1200000

Name: b, dtype: object

---Select 3rd row from a DataFrame---

name	Tammanah
Age	26

```

Profession      HR
Salary          1100000
Name: d, dtype: object

```

---Select 1 and 3 rows from a DataFrame---

```

      name Age Profession  Salary
b      Mit  32    Analyst 1200000
d Tammanah  26         HR 1100000

```

```

1  # Assignment: conditional loc-ing
2  import pandas as pd
3  import numpy as np
4
5  data = pd.DataFrame({
6      'Age' :      [ 10, 22, 13, 21, 12, 11, 17],
7      'Section' : [ 'A', 'B', 'C', 'B', 'B', 'A', 'A'],
8      'City' :      [ 'Gurgaon', 'Delhi', 'Mumbai', 'Delhi',
9                      'Mumbai', 'Delhi', 'Mumbai'],
10     'Gender' : [ 'M', 'F', 'F', 'M', 'M', 'M', 'F'],
11     'Favourite_Color' : [ 'red', np.NAN, 'yellow', np.NAN, 'black',
12                           'green', 'red']})
13
14 print(data)
15 print(data.iloc[1:3,2:4])
16 print(data.loc[data.Age >= 15])
17 print(data.loc[(data.Age >= 12) & (data.Gender == 'M')])
18 print(data.loc[(data.Age >= 12), ['City', 'Gender']])
19 data.loc[(data.Age >= 12), ['Section']] = 'M'
    print(data)

```

```

      Age Section  City Gender Favourite_Color
0     10      A  Gurgaon     M           red
1     22      B   Delhi     F           NaN
2     13      C  Mumbai     F        yellow
3     21      B   Delhi     M           NaN
4     12      B  Mumbai     M         black
5     11      A   Delhi     M         green
6     17      A  Mumbai     F           red

```

```

      City Gender
1   Delhi      F
2  Mumbai      F

```

```

      Age Section  City Gender Favourite_Color
1     22      B   Delhi     F           NaN
3     21      B   Delhi     M           NaN
6     17      A  Mumbai     F           red

```

```

      Age Section  City Gender Favourite_Color
3     21      B   Delhi     M           NaN
4     12      B  Mumbai     M         black

```


	City	Gender
1	Delhi	F
2	Mumbai	F
3	Delhi	M
4	Mumbai	M
6	Mumbai	F

	Age	Section	City	Gender	Favourite_Color
0	10	A	Gurgaon	M	red
1	22	M	Delhi	F	NaN
2	13	M	Mumbai	F	yellow
3	21	M	Delhi	M	NaN
4	12	M	Mumbai	M	black
5	11	A	Delhi	M	green
6	17	M	Mumbai	F	red

```

1  import pandas as pd
2
3  zoo = pd.read_csv('/home/aditya/Downloads/zoo.csv', delimiter = ',')
4  print(zoo)
5  print(zoo.count())
6  print(zoo.animal.count())
7  print(zoo.water_need.sum())
8  print(zoo.sum())
9  print(zoo.water_need.min())
10 print(zoo.water_need.max())
11 print(zoo.water_need.mean())
12 print(zoo.water_need.median())
13 print(zoo.groupby('animal').mean())
14 print(zoo.groupby('animal').mean().water_need)

```

	animal	uniq_id	water_need
0	elephant	1001	500
1	elephant	1002	600
2	elephant	1003	550
3	tiger	1004	300
4	tiger	1005	320
5	tiger	1006	330
6	tiger	1007	290
7	tiger	1008	310
8	zebra	1009	200
9	zebra	1010	220
10	zebra	1011	240
11	zebra	1012	230
12	zebra	1013	220
13	zebra	1014	100
14	zebra	1015	80
15	lion	1016	420

```

16      lion      1017      600
17      lion      1018      500
18      lion      1019      390
19  kangaroo      1020      410
20  kangaroo      1021      430
21  kangaroo      1022      410
animal      22
uniq_id      22
water_need      22
dtype: int64
22
7650
animal      elephantelephantelephanttigertigertigertigerti...
uniq_id      22253
water_need      7650
dtype: object
80
600
347.72727272727275
325.0

      uniq_id  water_need
animal
elephant    1002.0  550.000000
kangaroo    1021.0  416.666667
lion        1017.5  477.500000
tiger       1006.0  310.000000
zebra       1012.0  184.285714
animal
elephant    550.000000
kangaroo    416.666667
lion        477.500000
tiger       310.000000
zebra       184.285714
Name: water_need, dtype: float64

```

```

1  import pandas as pd
2  #Create a Dictionary of series
3  d = {'Name':pd.Series(['Sachin','Dhoni','Virat','Rohit','Shikhar']),
      ↪  'Age':pd.Series([26,25,25,24,31]),
      ↪  'Score':pd.Series([87,67,89,55,47])}
4  #Create a DataFrame
5  df = pd.DataFrame(d)
6  print("Dataframe contents without sorting")
7  print(df)
8  df=df.sort_values(by=['Age', 'Score'],ascending=[True,False])
9  print("Dataframe contents after sorting")

```

```
10 print (df)
```

Dataframe contents without sorting

	Name	Age	Score
0	Sachin	26	87
1	Dhoni	25	67
2	Virat	25	89
3	Rohit	24	55
4	Shikhar	31	47

Dataframe contents after sorting

	Name	Age	Score
3	Rohit	24	55
2	Virat	25	89
1	Dhoni	25	67
0	Sachin	26	87
4	Shikhar	31	47

```
1 import pandas as pd
2 import numpy as np
3 #Create a Dictionary of series
4 d =
  ↳ {'Name':pd.Series(['Sachin','Dhoni','Virat','Rohit','Shikhar']), 'Age':pd.Series([26,25,25,24,31]),
  ↳ 'Score':pd.Series([87,67,89,55,47])}
5 #Create a DataFrame
6 df = pd.DataFrame(d)
7 df=df.reindex([1,4,3,2,0])
8 print("Dataframe contents without sorting")
9 print (df)
10 df1=df.sort_index()
11 print("Dataframe contents after sorting")
12 print (df1)
```

Dataframe contents without sorting

	Name	Age	Score
1	Dhoni	25	67
4	Shikhar	31	47
3	Rohit	24	55
2	Virat	25	89
0	Sachin	26	87

Dataframe contents after sorting

	Name	Age	Score
0	Sachin	26	87
1	Dhoni	25	67
2	Virat	25	89
3	Rohit	24	55
4	Shikhar	31	47

```

1 import pandas as pd
2 import numpy as np
3 #Create a Dictionary of series
4 d = {'Name':pd.Series(['Sachin','Dhoni','Virat','Rohit','Shikhar']),
5      'Age':pd.Series([26,25,25,24,31]),
6      'Score':pd.Series([87,67,89,55,47])}
7 #Create a DataFrame
8 df = pd.DataFrame(d)
9 print("Dataframe contents")
10 print (df)
11 print(df.var())

```

Dataframe contents

	Name	Age	Score
0	Sachin	26	87
1	Dhoni	25	67
2	Virat	25	89
3	Rohit	24	55
4	Shikhar	31	47

Age 7.7
Score 352.0
dtype: float64

```

1 from collections import OrderedDict
2 from pandas import DataFrame
3 import pandas as pd
4 import numpy as np
5 table = OrderedDict((
6     ("ITEM", ['TV', 'TV', 'AC', 'AC']),
7     ('COMPANY', ['LG', 'VIDEOCON', 'LG', 'SONY']),
8     ('RUPEES', ['12000', '10000', '15000', '14000']),
9     ('USD', ['700', '650', '800', '750'])
10 ))
11 d = DataFrame(table)
12 print("DATA OF DATAFRAME")
13 print(d)
14 p = d.pivot(index='ITEM', columns='COMPANY', values='RUPEES')
15 print("\n\nDATA OF PIVOT")
16 print(p)
17 print (p[p.index=='TV'].LG.values)

```

DATA OF DATAFRAME

	ITEM	COMPANY	RUPEES	USD
0	TV	LG	12000	700

1	TV	VIDEOCON	10000	650
2	AC	LG	15000	800
3	AC	SONY	14000	750

DATA OF PIVOT

COMPANY	LG	SONY	VIDEOCON
ITEM			
AC	15000	14000	NaN
TV	12000	NaN	10000
['12000']			

4 Matplotlib

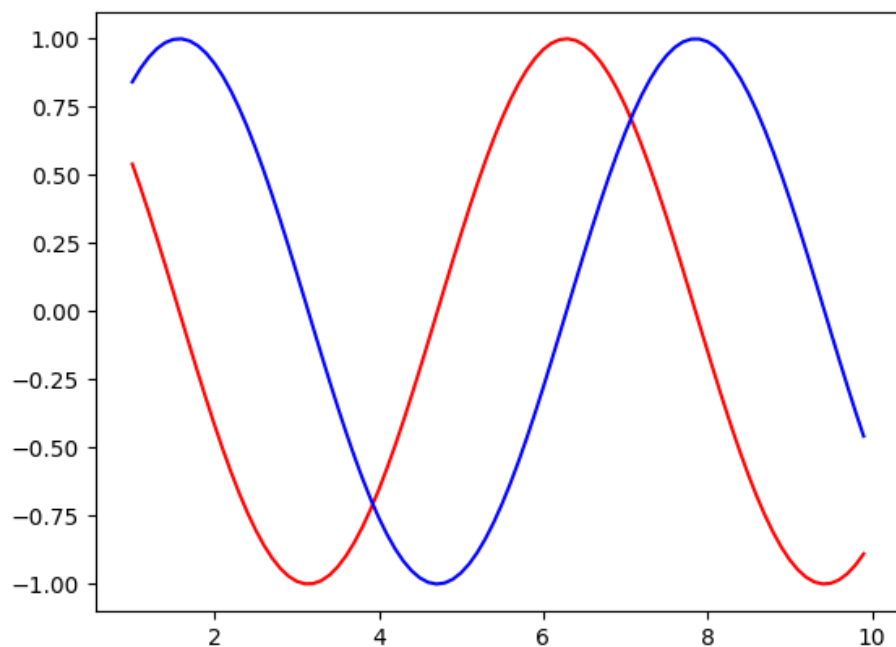
The next two blocks are the preamble and the postamble for all code blocks in `matplotlib`. These prevent repetitive code writing.

```
1 import matplotlib
2 matplotlib.use('Agg')
3 import matplotlib.pyplot as plt
4 import numpy as np
```

```
1 plt.savefig(path)
2 return path
```

4.1 Some Simple Plots

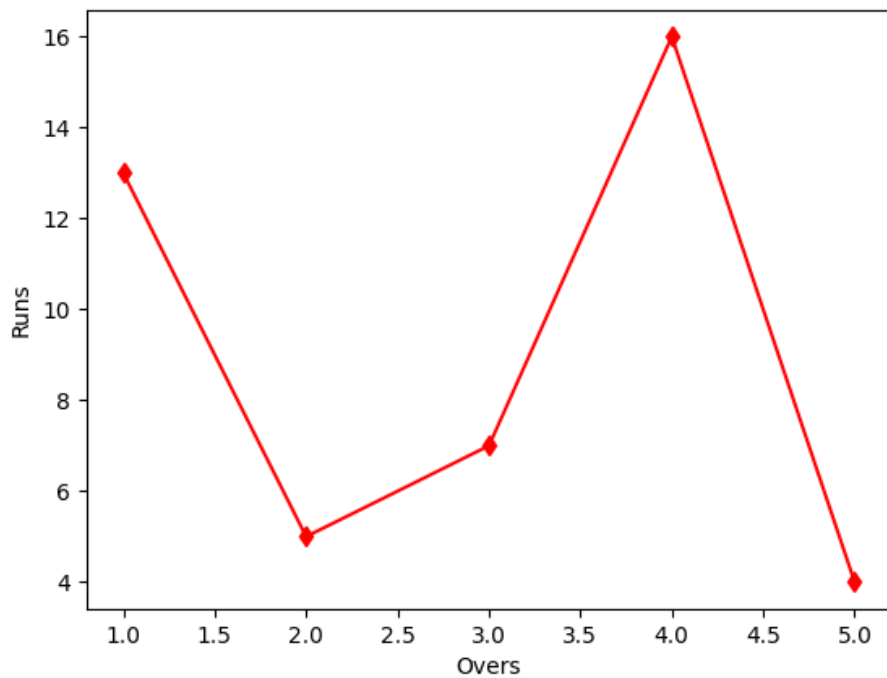
```
1 x = np.arange(1, 10, 0.1)
2 a = np.cos(x)
3 b = np.sin(x)
4 plt.plot(x, a, 'r')
5 plt.plot(x, b, 'b')
```



```

1  over = [1,2,3,4,5]
2  run = [13,5,7,16,4]
3  plt.xlabel("Overs")
4  plt.ylabel("Runs")
5  plt.plot(over, run, 'r', marker='d', markersize=6,
    ↪  markeredgecolor='red')

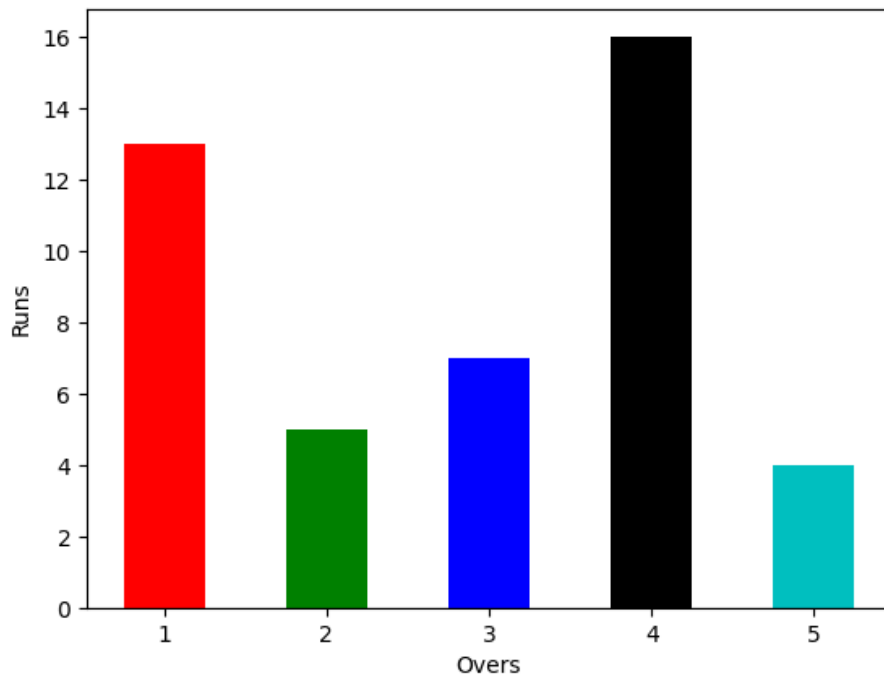
```



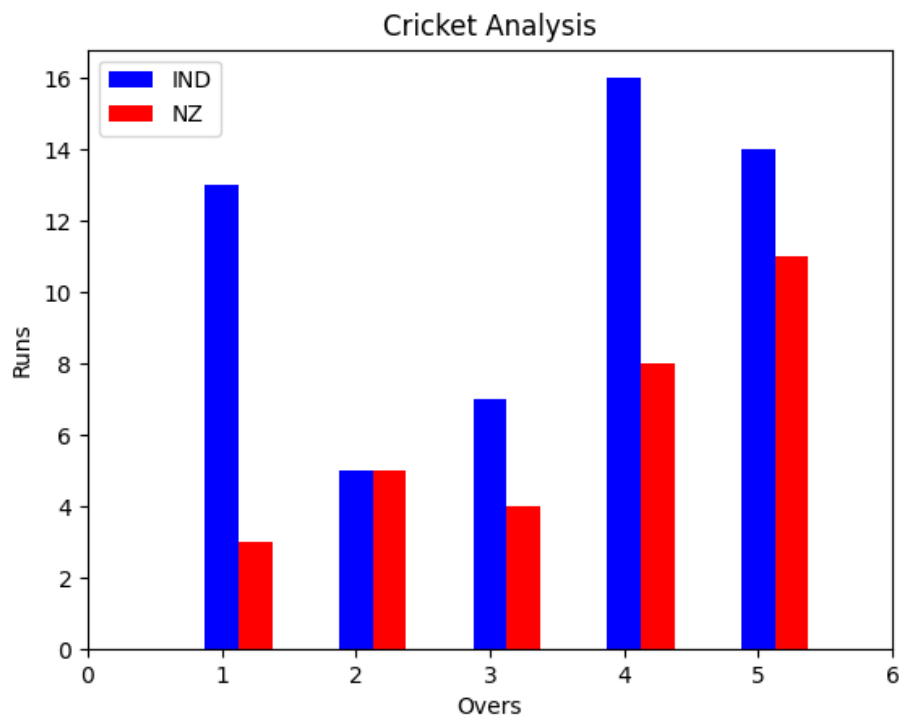
```

1  over = [1,2,3,4,5]
2  run = [13,5,7,16,4]
3  plt.xlabel("Overs")
4  plt.ylabel("Runs")
5  plt.bar(over, run, width=1/2, color = ['r', 'g', 'b', 'k', 'c'])
6  plt.show()

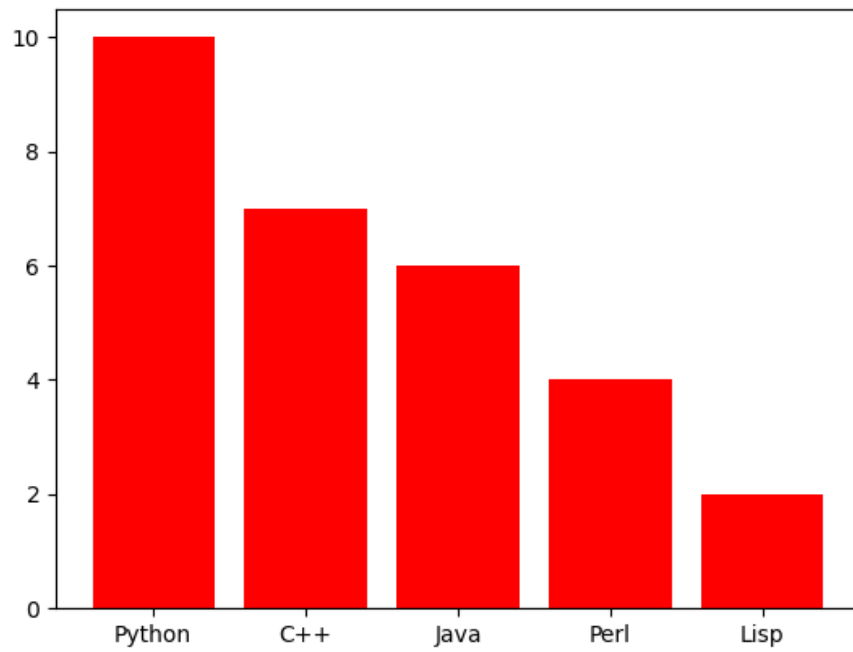
```



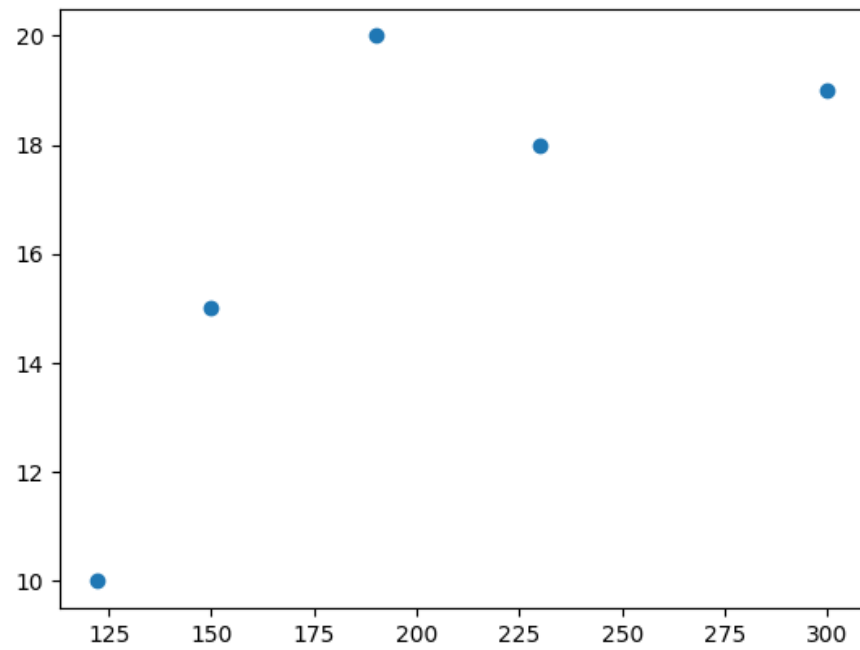
```
1 over = np.arange(1.0,6.0,1.0)
2 ind = [13,5,7,16,14]
3 nz = [3,5,4,8,11]
4 plt.xlabel("Overs")
5 plt.ylabel("Runs")
6 plt.xlim(0,6)
7 plt.title("Cricket Analysis")
8 plt.bar(over, ind, color='b', width=0.25, label = 'IND')
9 plt.bar(over+0.25, nz, color='r', width=0.25, label = 'NZ')
10 plt.legend(loc='upper left')
```

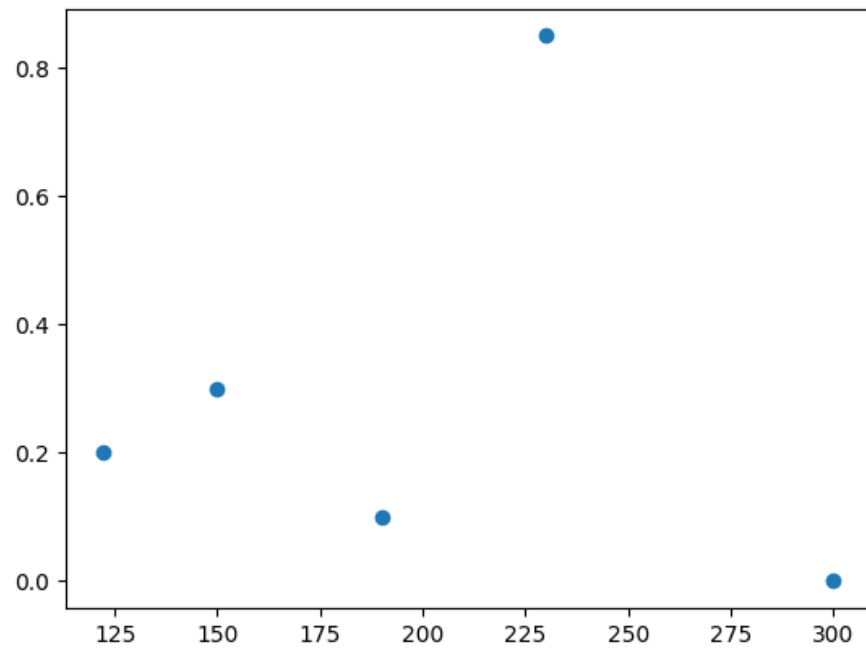
```
1 proglang=['Python', 'C++', 'Java', 'Perl', 'Lisp']
2 performance=[10,7,6,4,2]
3 plt.xlabel('Programming Languages')
4 plt.ylabel('Performance')
5 plt.bar(proglang,performance, color='red')
```



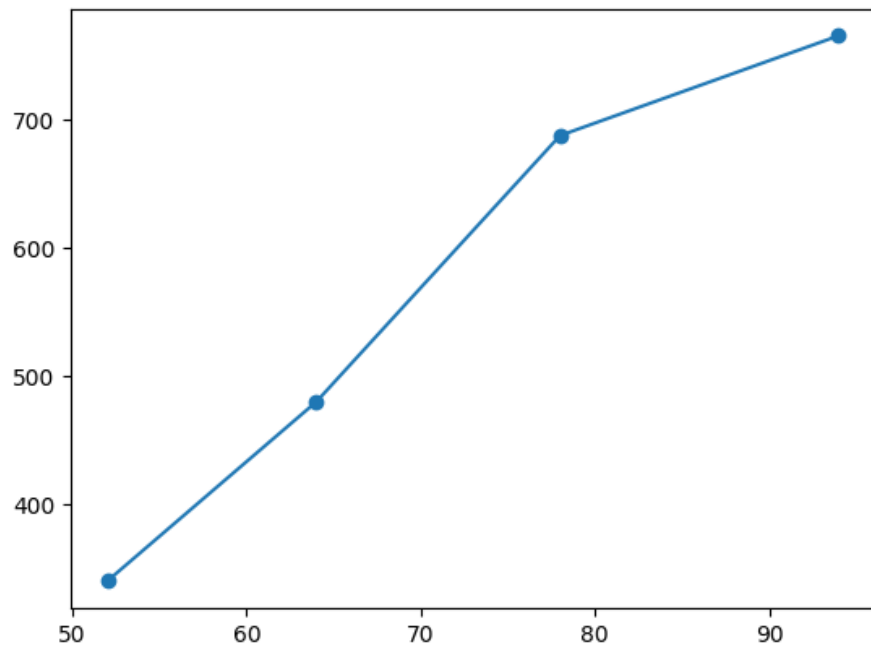
```
1 import pandas as pd
2 x = {'speed': [10, 15, 20, 18, 19], \
3      'meters': [122, 150, 190, 230, 300], \
4      'weight': [0.2, 0.3, 0.1, 0.85, 0.0]}
5 df = pd.DataFrame(x)
6 plt.scatter(list(df['meters']), list(df['speed']))
```



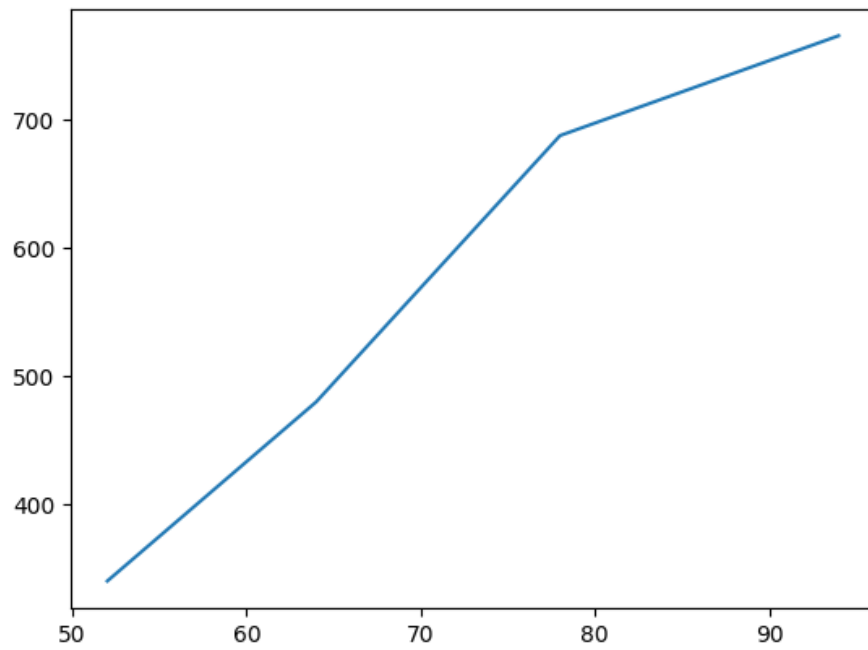
```
1 import pandas as pd
2 x = {'speed': [10, 15, 20, 18, 19], \
3      'meters': [122, 150, 190, 230, 300], \
4      'weight': [0.2, 0.3, 0.1, 0.85, 0.0]}
5 df = pd.DataFrame(x)
6 plt.scatter(list(df['meters']), list(df['weight']))
```



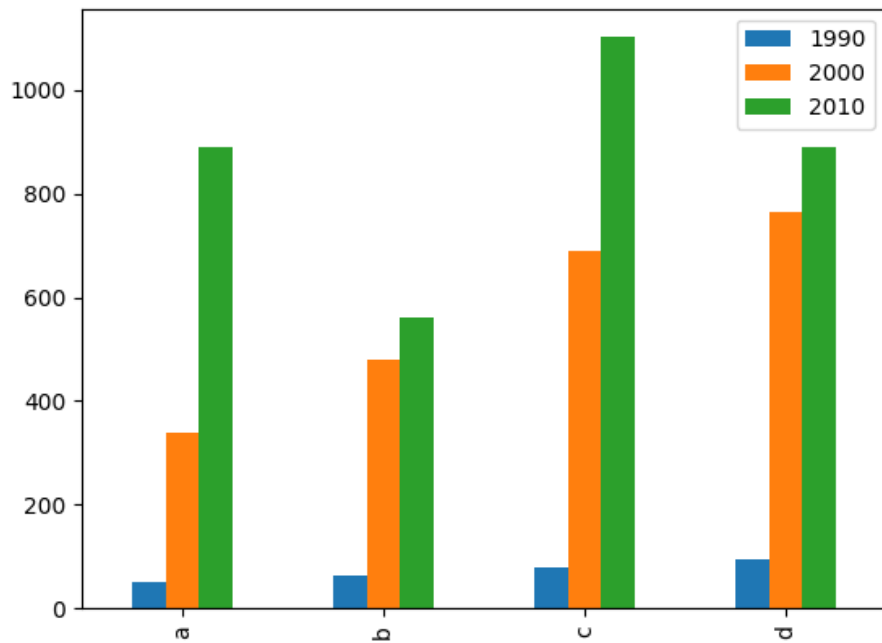
```
1 import pandas as pd
2 df1 = pd.DataFrame({'1990': [52, 64, 78, 94], '2000': [340, 480, 688, 766],
3   ↪ '2010': [890, 560, 1102, 889]})
4 df1.index = ['a', 'b', 'c', 'd']
5 plt.scatter(list(df1['1990']), list(df1['2000']))
6 plt.plot(df1['1990'], df1['2000'])
```



```
1 import pandas as pd
2 df1 = pd.DataFrame({'1990': [52, 64, 78, 94], '2000': [340, 480, 688, 766],
3   ↪ '2010': [890, 560, 1102, 889]})
4 df1.index=['a', 'b', 'c', 'd']
5 plt.plot(df1['1990'], df1['2000'])
```



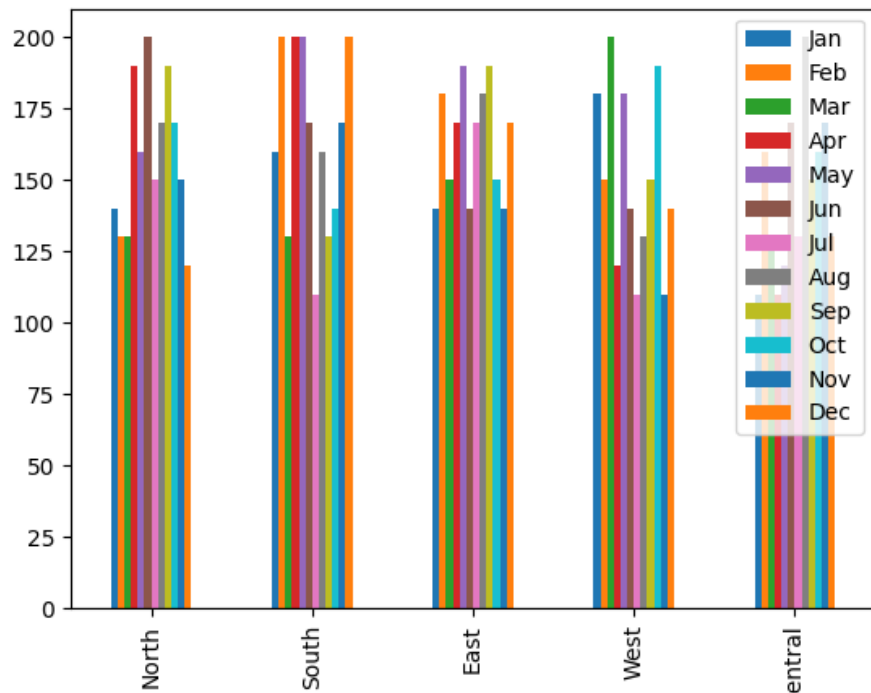
```
1 import pandas as pd
2 df1 = pd.DataFrame({'1990': [52, 64, 78, 94], '2000': [340, 480, 688, 766],
3   ↪ '2010': [890, 560, 1102, 889]})
4 df1.index=['a', 'b', 'c', 'd']
5 df1.plot.bar()
```



```

1 import pandas as pd
2 df1 = pd.DataFrame({'Jan': [140, 160, 140, 180, 110], 'Feb': [130, 200,
  ↳ 180, 150, 160], 'Mar': [130, 130, 150, 200, 130], 'Apr': [190, 200,
  ↳ 170, 120, 110], 'May': [160, 200, 190, 180, 120], 'Jun': [200, 170,
  ↳ 140, 140, 170], 'Jul': [150, 110, 170, 110, 130], 'Aug': [170, 160,
  ↳ 180, 130, 200], 'Sep': [190, 130, 190, 150, 150], 'Oct': [170, 140,
  ↳ 150, 190, 160], 'Nov': [150, 170, 140, 110, 170], 'Dec': [120, 200,
  ↳ 170, 140, 130]})
3 df1.index=['North', 'South', 'East', 'West', 'Central']
4 print(df1)
5 df1.plot.bar()

```



```

1 import pandas as pd
2 df1 = pd.DataFrame({'Jan': [140, 160, 140, 180, 110], 'Feb': [130, 200,
    ↪ 180, 150, 160], 'Mar': [130, 130, 150, 200, 130], 'Apr': [190, 200,
    ↪ 170, 120, 110], 'May': [160, 200, 190, 180, 120], 'Jun': [200, 170,
    ↪ 140, 140, 170], 'Jul': [150, 110, 170, 110, 130], 'Aug': [170, 160,
    ↪ 180, 130, 200], 'Sep': [190, 130, 190, 150, 150], 'Oct': [170, 140,
    ↪ 150, 190, 160], 'Nov': [150, 170, 140, 110, 170], 'Dec': [120, 200,
    ↪ 170, 140, 130]})
3 df1.index=['North', 'South', 'East', 'West', 'Central']
4 plt.plot(df1.columns.values.tolist(), df1.loc['North',:], label='North')
5 plt.plot(df1.columns.values.tolist(), df1.loc['South',:], label='South')
6 plt.plot(df1.columns.values.tolist(), df1.loc['East',:], label='East')
7 plt.plot(df1.columns.values.tolist(), df1.loc['West',:], label='West')
8 plt.plot(df1.columns.values.tolist(), df1.loc['Central',:],
    ↪ label='Central')
9 plt.legend()

```